Smart Contract Penetration Test Report

DApp Name: BulkSender

Test Date: 02-01-2025

Tester: Alex Cipher

Test Scope:

- **Contracts Tested**: BulkSender.sol, BulkSenderV2.sol, IBulkSender.sol, Proxies.sol, ERC1155.sol, ERC20.sol, ERC721.sol
- **Focus**: Security vulnerabilities and exploitation scenarios
- **Methodology**: Active penetration testing, manual review, and exploitation attempts

1. Executive Summary

Overall Risk: High

- **Critical Findings**: 1
- **Number of Findings**: 1
- **Test Conclusion**: High Risk

2. Penetration Test Findings

Critical Vulnerabilities

Issue: Unchecked if value sent is at least the intended value.

Risk: This vulnerability allows an attacker to exploit the contract, potentially draining all funds found in BulkSender.sol contract.

PoC:

In BulkSender.sol the function BulkTransfer() does not validate if user has sent enough funds to cover the cost of all value combined, therefore allowing for a malicious actor to make _values more than what he sent to the contract.

Recommendation: Implement a check to validate that the user deposits at least as much as the inputted _values.

Impact: Attackers can drain funds from the contract, negatively affecting all involved parties.

Status: Closed

3. Non-Security Recommendations for Usability and Performance

These are suggestions for enhancing user experience and overall DApp quality.

Usability Enhancements

- **Issue**: Lack of 0x0 Address Validation

Recommendation: Reject 0x0 addresses to prevent users from sending transactions to an invalid address.
Status: Open – Dev said: ***burning tokens is achieved by transferring them to the 0x0***

4. Testing Methodology

- The testing methodology for this engagement followed a structured approach to identify and exploit vulnerabilities effectively, ensuring comprehensive security coverage. The steps taken were:
- **Understanding the Contract**
    - The initial step involved reviewing the purpose and intended functionality of the DApp and its contracts. This allowed for a clear understanding of the business logic and how the smart contracts should behave, providing context for testing and vulnerability identification.

- **Understanding the Code Base**
    - A thorough analysis of the entire codebase was conducted, including all contract files, libraries, and dependencies. The objective was to identify interdependencies and potential areas of weakness within the code that could lead to security issues.
- **Using Static Analysis Tools (Slither)**
    - Static analysis was performed using **Slither**, an automated tool designed to identify potential vulnerabilities in the smart contract code. Slither helped pinpoint common security issues, such as reentrancy, improper access control, and uninitialized variables.
- **Manual Code Review**
    - Following the static analysis, a **manual code review** was performed to identify any vulnerabilities missed by automated tools. This step included inspecting the contract logic for potential attack vectors, analyzing external contract interactions, and ensuring that best practices were followed for security.
- **Testing Vulnerabilities in Hardhat Testnet**
    - Finally, identified vulnerabilities were tested in a **Hardhat testnet** environment to simulate real-world conditions and determine their exploitability. This involved running custom test scripts to validate the feasibility of attacks and assess the potential impact of the vulnerabilities in a controlled setting.

5. Conclusion

Overall, the DApp shows a **HIGH** risk profile due to the identified vulnerability. Immediate attention is required for the critical issue.

Next Steps:

- Address the critical issue as soon as possible to reduce the risk of exploits.
- Revisit the code after implementing fixes for a follow-up review.

Final Note: This report provides actionable insights into improving the security and usability of the DApp. Please feel free to reach out for further clarification or additional testing.